Easy Generation of Facial Animation using Motion Graphs

D. W. Fellner[†1,2] and S. Behnke[2]

[1]TU Darmstadt & Fraunhofer IGD, Germany
[2]Graz University of Technology, Institute of Computer Graphics and Knowledge Visualisation, Austria

Figure 1: *Example of unique animations generated automatically using our motion graphs technique from the same sequence of input expression labels, namely fear, surprise and happy. These animations are extracted from the accompanying video and have different poses and temporal dynamics.*

Abstract

Facial animation is a time consuming and cumbersome task that requires years of experience and/or a complex and expensive set-up. This becomes an issue especially when animating the multitude of secondary characters required e.g. in films or video-games. We address this problem with our novel technique that relies on motion graphs to represent a landmarked database. Separate graphs are created for different facial regions, allowing a reduced memory footprint compared to the original data. The common poses are identified using a Euclidean-based similarity metric and merged into the same node. While this process traditionally requires a manually chosen threshold, we simplify it by optimizing this value for the desired graph compression. Motion synthesis occurs by traversing the graph using the Dijkstra's algorithm, and coherent noise is introduced by swapping some path nodes with their neighbors. The expression labels, extracted from the database, provide the control mechanism for the animation. We present a way of creating facial animation with reduced input that automatically controls timing and pose detail. Also, our technique easily fits within video-game and crowd animation contexts, allowing the characters to be more expressive with less effort. Furthermore, it provides a starting point for content creators aiming to bring more life into their characters.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Animating a character's face is crucial to give it the illusion of a thinking brain. In films and video-games, the audience (or player) focuses mostly on main characters that naturally undertake most artistic attention. Animation of secondary characters, such as the ones seen in the street of game or in a crowd, does not have the same attention to detail due to cost and time constraints. Therefore, facial animation of secondary characters is reduced to a small array of animations/expressions that are constantly played. If the character is only seen once, this is not an issue, however when that is not the case, the repetitions become noticeable and may even hinder the immersion in the medium. Facial animations

† Chairman Eurographics Publications Board

are currently produced using either key-frame and performance-driven techniques, which despite producing very good quality results are either slow, or require special equipment and the additional actor. Procedural animation, i.e. motion generated by an algorithm, is an alternative capable of producing unique results with a low cost. However, most research has focused on body animation (e.g. [HG07,CTGH12,HTCH15]), leaving facial animation seldom studied. Algorithm based techniques require considerably less input, hence they are particularly suited for generating animations for the multitude of secondary characters present in video-games and in crowds of films.

In this paper, we present a novel end-to-end procedural facial animation system inspired by the motion graphs of Kovar et al. [KGP02]. The proposed method achieves unique and on-the-fly motion synthesis, controlled via a small number of parameters using a compact structure. The motion of each facial region is encoded in a separate motion graph obtained from the analysis of a labeled and landmarked facial motion DataBase (DB). Each graph is self-contained to avoid leakage, i.e. movement of one region influencing another. The graphs are created by comparing the poses in the DB using a similarity metric that accounts for the positions and velocities of the landmarks. When the value is below a certain threshold, the poses are merged in the same graph node. The author controls this threshold indirectly by specifying the desired graph compression ratio. Each time a sample is analysed and merged into the graph, the thresholds are individually optimized to match the chosen compression. Additional data is stored in order to reduce the information lost when merging multiple poses into a single node. Motion synthesis occurs by using Dijkstra's algorithm [Dij59] to find the path that minimizes the similarity between a source and sink/target node. The labels in the database allow finding the relevant target nodes, providing an intuitive control interface. The information contained in each path node is then used to recover the final animation. Uniqueness is achieved via a combination of noise in the path and independently calculated region paths. Finally, the generated motion is smoothed using the Savitzky-Golay window-based filter [Orf96] and sigmoid fitting to remove any jitter in the path. The main advantages of our method are:

- Compact representation the DB and intuitive control of the desired compression;
- Fast generation of animations on-the-fly near real-time;
- Unique animations in each generation;
- Authoring of animation with a low number of input parameters;

These advantages make the method suitable for generation of large quantities of facial animation, considerably reducing the cost of animating secondary characters. The remainder of this paper is organized as follows. We start by presenting the different approaches that use motion graphs in body animation, followed by the challenges of applying these to the face, sec. 2. In section 3, we present an overview of how the full method. In section 4, we focus only on the creation of the different motion graphs from the analysis of a motion database. The next section deals with extraction and recovery of the animation from the graph, and the process of introducing coherent noise, sec. 5. In section 6, we present the results, compare them with [ZSCS04], and discuss the limitations and factors that influence the results. Finally, the conclusion and future work are presented in section 7.

2. Related Work

Procedural animation approaches are different from performance-driven and key-frame techniques since they are capable of generating an animation from discrete input. The author pre-configures an algorithm which then generates the animation from discrete parameters or events, effectively controlling the animation without directly manipulating the model's transformation/deformation. Procedural animation techniques can be loosely divided into 3 classes: *constraint* or *rule*-based [PG96, Per97, BHPN01], where rules impose limits and variations on the generated motion; *statistical* or *knowledge*-based [KGP02, HG07, SBR*14, HTCH15], with the motions learnt from examples; and finally *behavior*-based, where the cognitive/emotional process is emulated [XMLD07]. In the last, the character's behavior is modeled to decide the animation, while statistical and constraint-based methods focus on the actual motion. These still require an author to control the parameters that generate small animation sequences, which does not occur in behavioral techniques. Hybrid techniques also exist and include [AD07, BSG10, SBR*14].

Motion graphs fit within *statistical* procedural animation, and have been widely used for body animation [KGP02, HG07, CTGH12]. Kovar et al. [KGP02] defined motion graphs as a directed graph capable of encoding motion data in a way that synthesising movements is done by traversing the graph. The movements can be encoded in the nodes or the edges. Motion graphs are different from move trees [MBC01] as the latter is created manually, while the former is automatic [KGP02]. Using motion graphs entails several aspects: defining the structure, creating the graph, choosing the path and handling the transitions. Different **structures** include using the edges to store small motion clips [KGP02] or place groups of animations in the nodes, i.e. parametric graphs [HG07, CTGH12]. The latter makes path selection faster, allowing for interactive rates [HG07], at the cost of more user interaction. **Graph creation** usually consists of comparing all the frames [KGP02] or nodes [HG07] and deciding, based on similarity metric, if they should be connected or merged. These nodes, traditionally, have information on the model's joints, however more recently [HTCH15] stored actual meshes. Kovar et al. [KGP02] used a Euclidean distance based metric, which considers frames around the pair being analysed. With a poor similarity metric, the animation will jitter, thus producing disconnected results. **Choosing the path** depends on the desired motion, [KGP02] chose the nodes that minimize a pre-defined route. Heck and Gleicher [HG07] presented approaches that: 1) keep the character moving to a location, 2) best follow user requests, e.g. direction. **Calculating transitions** between nodes is also important, as each node can have content from different samples and directly concatenation can lead to jitter. [KGP02] found the 2D transformation that converts joints positions from one edge to another. Gleicher et al. [GSKJ03] used a displacement map to improve the smoothness. Casas et al. [CTGH12] minimized a similarity metric based on the shape, motion and latency. Parametric approaches face a more delicate problem as each node generates a large amount of animations, e.g. [HG07] sampled the parameter space and chose the best candidates. For more details on motion graphs, the reader is referred to [Gle08]

The application of motion graphs to facial animation is scarce

compared to body animation. As Orvalho et al. [OBP*12] points out, facial movements are more complex than body movements. Although both are temporally non-linear, facial movements are also non-linear in terms of shape. Both have issues with rigid alignment, the body due to different orientations, and the face due to head movements. Most work on body motion synthesis has focused on tasks like locomotion or grabbing objects/fighting, where the constraints used, such as foot placement, are well understood. This does not occur with non-verbal behaviors in both body and facial motion synthesis. Besides, there is a considerable amount of body motion data available to train procedural approaches, while facial data is substantially scarcer. The only known exception is the work of Zhang et al. [ZSCS04], which relies in a graph created by connecting all poses in the training sequence, and each node contains a facial mesh. The user specifies the source and destination nodes, and the path is chosen by minimizing a L2-norm based similarity metric. Variations are added by traversing the graph several times for different parts of the face. The authors also introduced the creation of a graph per facial region, with each used to find the optimal in-betweens of manually defined key-poses. Our approach shares some concepts with Zhang et al. [ZSCS04]. However, we extract meaningful connections from the samples instead of connecting all nodes, which reduces the graph size considerably. Similar to [ZSCS04], we also traverse each region graph. However, we vary the chosen paths for coherent noise. Finally, expression labels facilitate specifying the nodes, thus easing the control of the animation.

As for procedural facial motion, Arya and DiPaola [AD07] presented an approach that uses 4 independent spaces: knowledge, personality, mood and geometry. The first is controlled via a XML-based language, the second and third have psychology bases, while the last relies on a hierarchy to define the motions. Another hierarchical approach was presented by Perlin [Per97], where the lowest level relies coherent noise applied to joints' motion, while the highest specifies an emotion that is blended with the current pose. Bidarra et al. [BSG10] modeled the character's internal state using the PAD model [MR76], with emotions, mood and personality mapped to points in this space. The mood is updated using a push and pull approach, which drives the choice of short animations placed in the PAD space, and blended according to the mood proximity. Xue et al. [XMLD07] presented an approach that uses fuzzy logic to combine existing expressions using parameters from 3 layers: social, emotional and physiological. While the expression is generated by blending, the timing needs to be manually controlled. Sagar et al. [SBR*14] combined statistical and behavioral models to create a generative model of facial expressions. They simulated, using e.g. recurrent neural network models, the different neurobiological systems that trigger the facial muscles. The animations are generated by activating precomputed biomechanically simulated deformations that are associated to a physically based model.

Most approaches [DDLT02] rely on some sort of hierarchy to control the animation [Per97, AD07, XMLD07, BSG10], which means there is an associated configuration cost. Motion graphs can significantly reduce this step, since they are automatically created and encode both timing and pose information. Sagar et al. [SBR*14] presented, arguably, the most advanced work on facial animation generation, however this leads to a complex set-up, specially for behaviors not previously learned by the model. Therefore, the novelties of our method are:

- Motion graphs based approach specifically tuned for facial animation that achieves a compact representation of the original motion. This requires a new node structure for encoding the original motion in a smaller space;
- Novel approach to ease the choice of the thresholds based on the desired compression. Most methods leave this choice to the user, which is a slow and challenging task as the threshold is a heuristic value that dependents on the similarity metric. Varying the threshold also tends to produce predictable results only with large variations. Optimizing the thresholds for the amount of information to keep in the graph considerably simplifies the fine tuning of a graph. Additionally, a threshold affects merging of all poses and samples equally, which is not ideal given that different sequences might have more subtle pose variations than others. Therefore, a threshold for one sequence might be too aggressive for another. Individual thresholds create graphs that represent the DB motions better and lose less information than with a global threshold;
- New method for introducing coherent noise in animation, by varying the nodes in the path;

3. Method Pipeline

For a better readability, we now present a small list of the most commonly used terms to refer to different aspects of our motion graph-based technique.

- *Region Graphs*: Motion graphs created for each facial area;
- *Sample/Sequence*: Sequence of poses/frames in the original DB;
- *Sample Graph*: Group of region graphs created from one sample, G_{samp};
- *Final Graph*: Group of region graphs that results from merging all sample graphs, G_f;
- *Expression labels*: Labels included in the DB samples. In the present case they include the basic emotions.
- *Node*: A graph node contains the landmarks displacement and additional data (sec. 4.4) used to reconstruct the motion (sec. 5.1);
- *Landmark displacement*: Difference between current landmarks coordinates and the base pose landmark coordinates. The base pose is the average of first frame/pose of all samples;
- *Source Node*: First node in a motion generation;
- *Destination/Sink Node*: Last node in a motion generation;
- *Compression*: % calculated based on the data difference between a graph with a node per sample pose and the current final graph.

Our approach for motion synthesis relies on first creating the region graphs from the analysis of the DB and second, traversing these structures to synthesize the motion, according to user input. This process is outlined in fig. 2. Motion Graph creation (sec. 4.1) consists in comparing all poses, via a similarity metric (sec. 4.2), from each DB sample to create a sample graph. The poses are merged into a single node when the similarity is below a certain threshold. All the sample graphs are then merged to create the final region graphs, using the same compare-and-merge approach, but in this case between nodes of the sample graphs. The thresholds,

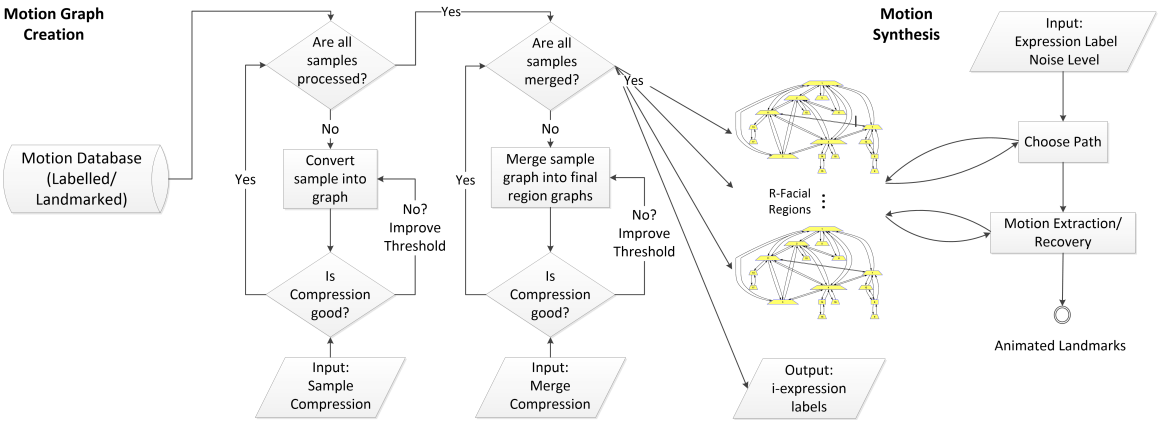

Figure 2: *Overview of the procedural animation pipeline. The DB is analyzed to create a graph per facial region. As the samples are analysed to create the graphs and then merged, their compression values are compared against the desired compression, and, if the value differs, the process is repeated with another threshold. Motion synthesis occurs by using the input expression label to find a path in the graph that allows recovering the final motion.)*

used when creating sample graphs and when merging them, are automatically determined via optimization for a desired compression (sec. 4.3). The author controls motion synthesis by using the same expression labels as the ones associated to DB samples. These labels are used to determine the next destination node, which then allows finding the minimum distance path that connects the current displayed facial expression and the next (sec. 5). The in-betweens and timing of the final motion are then obtained by analysing the path nodes (sec. 5.1). Coherent noise is introduced by varying the path to generate unique sequences (sec. 5.2).

4. Motion Graph Generation

The creation of the region motion graphs starts with a dynamic 2D/3D sparse DB, whose samples need to be aligned to remove influences of identity and head movements. The choice of the DB followed the requirements of having: sparse landmarks, expression labels, multiple samples per label and most intense poses identified, i.e. peak-poses. While dense data provides more information, sparsity is more lightweight and considerably reduces leakage of motion from one facial region to another. We selected the Cohn-Kanade (CK and CK+) data set [KCT00,LCK*10] because it fits all the requirements in spite of not being originally intended for motion synthesis. Alternatives include [ZYC*14, LCP*11], however the first is not free and the second does not have enough expression labels. CK/CK+ expression labels are associated to the emotions of: *happiness, sadness, disgust, surprise, anger, fear, contempt* plus *neutral*. All samples follow the structure: neutral-to-peak expression, with each pose containing 68 landmarks. CK/CK+ requires pre-processing to reduce errors/jitter and to remove the identity and head movements (important for accurate similarity values). Therefore, we clean the data manually, and then fit a sigmoid to each landmark displacement, per sample. This process smooths the motion while keeping its individual variations and timings. The sigmoid function (defined in eq. 1) was chosen as it incorporates the acceleration variations of human motion. The sigmoid is controlled by: b_1, curve's maximum value; b_2, steepness; and b_3 translation. Least squares (eq. 2) is employed to find the optimal parameters for each sample data, with m being the number of points in the curve.

$$g_{b_1,b_2,b_3} = \frac{b_1}{1+e^{-b_2(x-b_3)}} \quad (1)$$

$$\min_{b_1,b_2,b_3} \sum_{i=1}^{m} \|y_i - g_{b_1,b_2,b_3}(x_i)\|^2 \quad (2)$$

Finally, the number of landmarks is reduced (fig. 3) to lower the overhead of graph creation, motion synthesis and errors in similarity calculation. Pose alignment is done using Procrustes analysis in two stages. First, for each sample, all poses are rigidly aligned with their first (neutral) pose using landmarks of nose and eyes corners. Secondly, for each sample and each region, the first pose is aligned with the average neutral expression of all the samples, and this transformation is applied to all sample poses. This results in *"non-rigid"* alignment where the regions are individually aligned, thus reducing the effect of different proportions. The landmarks used in this second stage can be seen in fig. 3.

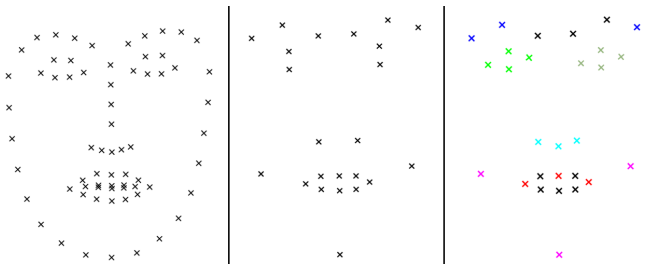

Figure 3: *On the left, the original 68 landmarks; In the middle, the reduced set used to create the graph; On the right, markers used for "non-rigid" alignment of the face regions, grouped by colors: blue for eyebrows; different greens for eyes, cyan for nose, red for mouth and magenta for jaw and cheeks*

The proposed method relies in region graphs instead of the holis-

tic approach with only one graph. This allows more accurate similarity values where the motion of one region does not affect the similarity value of another. It also leads to more compression, as each graph only stores its most significant motions. Additionally, the holistic approach makes the thresholds considerably harder to control, and would require for a finer grained optimization (sec. 4.3). The regions chosen for this paper are: eyes, eyebrows, nose, mouth, with the cheeks and jaw grouped into another. This choice is based on observation of the samples' movements, although other configurations are possible.

4.1. Graph Creation

Graph creation is the same irrespective of the region. It first converts each sample into a sample graph, G_{samp}. The connections between adjacent nodes/poses are established here, hence storing the temporal dynamics. As a result, the order in which the poses occur becomes part of the sample graph in the form of edges. Afterwards, the sample graphs are merged to create the final graphs, G_f (shown in fig. 4.) Similar nodes are identified, and serve as transition points, effectively bridging one sample to another. As both stages are similar, we now describe the process of creating a sample graph (algorithm 1). The differences between stages are highlighted afterwards. Calculating the similarity between two poses/nodes and finding the optimal thresholds are described in the following sections.

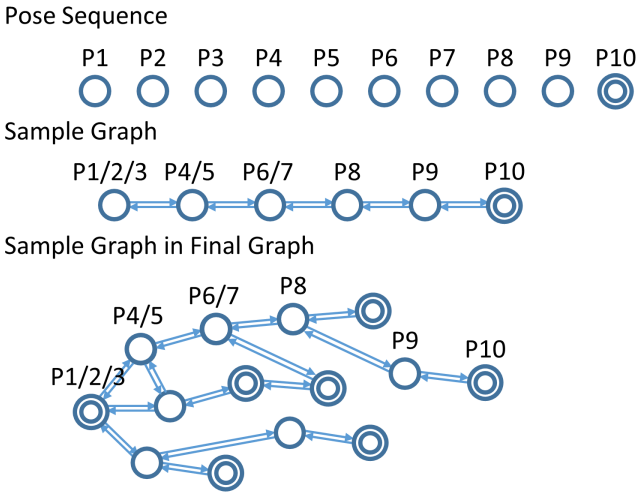

Figure 4: *Example of what happens to a sample through the whole graph creation process (only shown for one region graph). The sequence is first converted to a sample graph and is then merged into the final graph. Sample nodes containing poses P1 to P8 are all merged into existing final graph nodes. Double circle nodes represent destination nodes, and contain peak expressions.*

At this point, it should be noted that all DB sequences need to start with the same/equivalent pose, P_{eq}, which is neutral in the case of CK/CK+. While how each sample unfolds is not relevant, this pose is the common denominator between all samples, allowing the landmark displacement to be calculated. This displacement is required for the similarity computation (sec. 4.4 for the reasoning).

We additionally average the landmarks of the equivalent pose of all the samples, $\overline{P_{eq}}$. A sample motion graph is then generated following these steps for each pose: **1)** calculate the displacement, D_{cur}, between current pose landmarks, P_{cur}, and the sample's first pose P_{1st}; **2)** add D_{cur} to $\overline{P_{eq}}$ to create P_{new}; This converts the sample's displacement/landmarks into their counterparts in the final graph. The P_{new} is used to **3)** calculate the similarity to all the current sample-graph nodes, $< S1...S_n >$ and choose the lowest value, S_{low}; if **4)** S_{low} is below the threshold, merge P_{new} with the lowest similarity node N_{low}; Otherwise, **5)** create a new node, N_{new}, and append it to the sample graph,. In both cases, a connection is established to the previous iteration node.

Algorithm 1 Sample Motion Graph Creation

```
1:  procedure CreateSequenceSubGraph
2:      Avg all equivalent sample poses >> P_eq (bar)
3:      for all sample frames do
4:          P_cur − P_1st >> D_cur
5:          D_cur + P_eq (bar) >> P_new
6:          CalcSimilarity(P_new, G_samp) >>< S1...S_n >
7:          ChooseLowestValueNode(< S1...S_n >) >> S_low, N_low
8:          if S_low < threshold then
9:              Merge P_new with N_low
10:         else
11:             Create node w/ P_new and D_cur >> N_new
12:             Add N_new to G_samp
13:         Connect N_low/N_new to prev. analysed node
```

Creating the final graph is an iterative process seeded with a random sample graph. Merging a sample graph with the final graph again follows the same comparing-and-merge approach. However, when the nodes are merged, we additionally use Procrustes Analysis to align the sample graph node with the absorbing final graph node using the same groups as in fig. 3. Uniform scaling is not considered to reduce changes in the shape. We additionally apply this transformation to all following sample nodes, since this will generally improve the similarity calculations when these are compared after the current merging. As the alignment is applied to the sample, there is no need to store or compute the transformations when synthesising motion.

4.2. Similarity Metric

The chosen similarity metric, eq. 3, takes into account both spatial location of the landmarks and their instantaneous velocity, with lower values representing higher similarity. When comparing two different poses a and b, we define that each pose/frame P_a is composed by n number of landmarks $< PaL_1...PaL_n >$. Thereby, the distance between two poses for the same landmark i is given by $dist(PaL_i, PbL_i) = \sqrt{\sum_{j=1}^{dimensions}(PaL_{i,j} - PbL_{i,j})^2}$. The instantaneous velocity of a landmark i for the P_a is given by $\vec{V}(PaL_i) = PaL_i - Pa_{prev}L_i$, where $Pa_{prev}L_i$ is the position of L_i in the pose/frame immediately before P_a. On top of this, we calculate the velocity's influence, v_{infl}, which is a scalar that represents how similar are the velocities of a landmark i in two poses. This is given by $v_{infl}(PaL_i, PbL_i) = 1 - |\vec{V}(PaL_i) \cdot \vec{V}(PbL_i)|$. v_{infl} varies between $[0,..,1]$: 0, if the two vectors are the same, independently

of the direction; 1, if they form $+/-90°$. If both vectors are close to opposite, we argue they represent onset or offset phases, thus the influence should be the same. This is achieved using the absolute value. The final similarity value is obtained via:

$$sim(P_a, P_b) = \sum_{i=1}^{n} dist(PaL_i, PbL_i)(1 + \lambda v_{infl}(PaL_i, PbL_i)) \quad (3)$$

λ controls the influence of the velocity (smoothness) in the result. For the implementation we have set the λ to 1. When this metric is computed between two nodes in graph(s), the instantaneous velocity is calculated for each incoming edge, and the lowest similarity is chosen as the representative value. The similarity metric is used both when identifying the similar nodes to be merged and to set the graph edge weights. These weights are only obtained on the finalised graph, due to the similarity metric requiring all incoming velocities of each node.

4.3. Optimizing for Compression

The author controls the graph by specifying the desired compression and the acceptable margin, e.g. 90% compression with 10% of tolerance, which are used to compare the number of nodes present after each stage. It is by varying these values that the author controls the trade-off between motion quality (smoothness) and flexibility of the graph (compactness). Lower compression values lead to more information being kept, since less nodes are merged. The different thresholds are independently optimized for the stages of sample graph creation and sample graph merging. When creating the sample graph, the number of sample nodes is compared with the original number of sample poses. When merging, the number of nodes that increase in the final graph is compared with the sample graph nodes. This determines whether the desired compression has been achieved or, if not, choose how to update the threshold. If the latter, the stage is repeated until the compression is obtained or it is no longer possible to update the threshold, since we follow discrete values. A discrete space is sufficient to discriminate between the poses given the used similarity metric and the database ranges. While the method could also work in a continuous space, via an additional ending condition, it is not guaranteed there exists an optimal threshold that achieves strictly the desired compression. This happens because a small change in the threshold can lead to several nodes being merged. The downside of this approach is that, if a very small tolerance is specified, the compression may not be met. Nevertheless, as each optimization approximates the defined goals, the final compression approximates the desired values. The presented method locally optimizes the thresholds for each stage/sequence, with the goal of achieving a global compression. This prevents the case where a threshold is too aggressive for one sequence but too lenient in another, which is the case of global threshold methods. Still, it is possible to configure our method to work with a global threshold, by setting the compression to 50% and tolerance to 50%. This means the initial threshold is used and any comparison between the number of nodes is valid.

On top of the compression goal, the author needs to provide an initial threshold and how it will initially vary, i.e. a step. Therefore, the main question is: how is the step updated? This process is based on the obtained compression, where if it is lower than desired, the threshold value is increased by the step. A higher threshold means more nodes will be merged, hence the compression will increase. If the compression is higher than the desired, then the number of nodes has been reduced too much. Hence, the threshold is decreased. The step itself will remain the same while the addition or subtraction operation can be repeated. After there is an change of the operation, each time the step is added or subtracted, it is divided by two. This will continue until the desired compression is obtained or the step is 1.

4.4. Graph Structure

The core structure of our approach is a directed graph, where each node contains the average landmarks displacement of all poses merged in it and each edge has a similarity value. The displacement was chosen, as opposed to the landmarks positions, to mitigate errors in alignment and effects of each person's proportions. The neutral pose of two people has different landmark positions but same displacement values. The first node created in the final graph additionally contains the average landmarks of all DB neutral poses, $\overline{P_{eq}}$. This results from the first node of all the samples being always associated to a displacement of 0, thus containing the base pose. This information is added to the displacement of a given node to create the actual coordinates of the landmarks. Each graph node also contains all expression labels from the sequences whose nodes were merged in it. We store additional information in the nodes to allow for better motion recovery (sec. 5.1). This information is calculated from all the poses merged in each node.

- Average number of consecutive merged nodes and their respective landmarks velocities. Consecutive refers to, when merging a sample graph into the final graph, a node from the latter might absorb consecutive n sample graph nodes. Both n and the respective landmarks velocities are stored. After the graph is created, these values are averaged;
- For destination nodes, i.e. nodes with at least one peak expression, we store the average number of frames from neutral-to-peak expressions and respective standard deviation;

5. Motion Synthesis

Generating new motion (as shown in fig. 5) now becomes a task of traversing the graph, choosing the nodes relevant to the desired facial behavior. The author can directly choose both the sources and sinks/destinations in all graphs, which provides very fine control. However, this approach requires deep knowledge of the graph. As an alternative, we assume the nodes containing peak, or apex, expressions, previously identified in DB samples, are the most desirable to animate a character. These nodes are flagged and associated to the respective expression label. By specifying the label, we control the next sink by randomly sampling a valid peak node. In a long animation, the current sink node becomes the source for the next iteration. The first source of the sequence is manually specified, which can be easily done if the pose is a peak or neutral node. The path is chosen using Dijkstra's algorithm [Dij59] that minimizes the similarity values between the source and sink nodes. This is done for all region graphs to create a full facial behavior. While the path is the basis for motion, the actual landmark movements still need to be extracted (sec. 5.1). The path cannot be used directly as the temporal dynamics were partially lost, e.g. a sequence with 10 poses

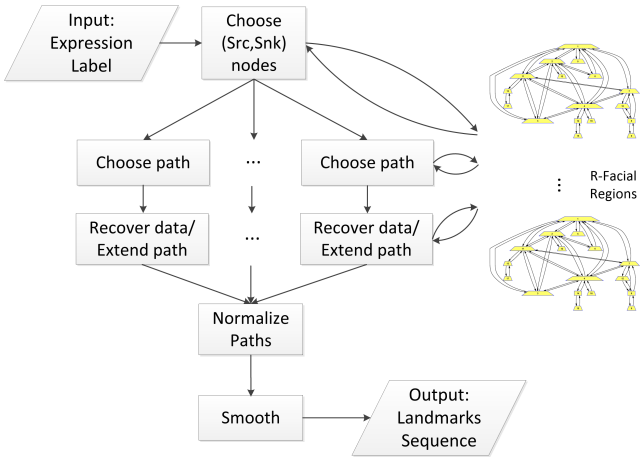

Figure 5: *Overview of the synthesis process. Expression labels allow finding the desired sinks in each graph, with motions generated by path finding. Each node's information is used to recover the landmarks positions. After, all the region motions are assembled and smoothed to obtain the final animation.)*

might be represented with only 3 nodes. Additionally, creating animations for multiple characters requires uniqueness. Therefore, we explore the structure of the graphs to add this extra variation (sec. 5.2).

5.1. Reconstructing motion from path

The data contained in each node (sec. 4.4) allows partially recovering the information lost when the poses were merged. This process starts with the Dijkstra's path nodes from where the core poses and dynamics are determined (step 1). Afterwards, these are extended to achieve a more realistic motion (steps 2 and 3) and finally smoothing removes any existing jitter (step 4). These steps rely primarily on basic operations that require a small amount of data, which keeps the stored data to a minimum.

1. Control the number of poses the node generates using the average of consecutive merged nodes, and extract each pose displacement from the average landmarks' velocity. This is crucial to adequately recover non-linear aspects of motion. Applying velocity depends on the number of poses. Only one pose means no velocity is applied, since the pose is represented by the node's displacement. With more poses, the node's displacement is used as the central point to which either the velocity, or its inverse, is added. This happens as poses are respectively created in the direction of the next or the previous node. Poses created this way are clamped to the poses present in the previous and next nodes, e.g. in a middle node of a neutral to smile motion, a smile created from the velocity application cannot be wider than the smile present in the following node. Nevertheless, this is an approximation that will produce linear motions in the limit, which is the case with a desired compression too high. This leads the graph to no longer represent the dynamics of facial behaviors.
2. Use the peak nodes average duration as the sequence length. The current displacement is stretched or shrunk accordingly using linear interpolation. While the previous step recovers the shape of the temporal dynamics, the final duration is usually too short as the nodes contain information from several samples. This allows a more realistic synthesis particularly in neutral-to-peak animations. Peak-to-peak transitions are also important, as such, we estimate their duration using the node of the path closest to the first graph node as a reference. The number of nodes between this node and both, the source and sink, are used as weighting factors of the respective peak durations. As an example, for a path of 10 nodes, if the closest node is the third starting from the source, they weights of the respective peak durations will be 0.3 and 07.
3. Normalize the displacements of each facial region. As each path is created independently, the lengths of the region paths tend to be different. The motions are normalized, via linear interpolation, by finding the longest and stretching the others to match its size. Linear interpolation is used here and in the previous step, however only for the in-betweens of the motion curve formed in the step 1. As a result, the motion as a whole does not become linear and keeps its core motion properties.
4. Smooth the sequences using the Savitzky-Golay window-based filter [Orf96] and sigmoid fitting (sec. 4). These approaches complement each other, as the first removes drastic motions, preventing the sigmoid to be too sharp, and the second removes any left zigzag. This tends to occur when generating sequences not in the DB or when path noise is introduced (sec. 5.2). Sigmoid curves are associated to motions with slow in and out, which is the case of most movements of facial regions. However, it can approximate linear motions, such as blinking, as long as the compression does not make the blinking nodes merge with other labels nodes;

5.2. Introducing Variation

Synthesizing animations that share a common label, but are slightly different, is a crucial requirement for our method. It allows creating an animation of a crowd that can react, e.g. to a joke, with the same core emotion but slightly different and idiosyncratic motions. Our approach inherently introduces noise, as each facial region has its own motion graph and respective path. Thus, a new sequence is composed by movements originated from different samples. We force additional variations in the sequences length and the chosen path. The first, relies on the standard deviation, stored per peak node, to define a normal distribution. We randomly sample this space and stretch/shrink the displacement accordingly. The path variations build on top of Dijkstra's path (sec. 5) by randomly replacing path nodes with their neighbours. These connect to the original path nodes and, as such, contain poses similar to the ones being replaced. To reduce the chance of selecting a completely different pose, we only consider neighbors that share at least one label with the current source or sink nodes. Noise is controlled by the percentage of the original path that can be changed and the number of neighbor hops from which a node can be selected. Thus, coherent noise is added without disrupting the whole sequence. Nevertheless, depending on the graph, the resemblance of nodes might be broken with more than 2 hops, making the noise random. With these two approaches, enough samples in the DB, and the inherent

Generation Time (seconds)	Input Compress. Seq. Creation (%)	Input Compress. Seq. Merge (%)	Nodes Compress. (%)	Data Compress. (%)	Seq. Duration +/- Deviation (fps)	Landmark Diff. +/- Deviation (pixels)	Angle Vel. Diff. +/- Deviation (%)	Synth. Times +/- Deviation (seconds)
3411,1	0,1+/-0,1	0,2+/-0,2	-54.07	22.74	0+/-0	1,43+/-1,96	14,49+/-13,49	0,39+/-0,45
5560,3	0,1+/-0,1	0,3+/-0,2	-10,27	44,22	0+/-0	1,67+/-1,967	14,85+/-13,77	0,49+/-0,60
4973.2	0,1+/-0,1	0,4+/-0,2	8,37	53,02	0+/-0	1,84+/-2,17	15,06+/-13,96	0,55+/-0,57
2859,6	0,2+/-0,1	0,5+/-0,3	38	66,8	0+/-0	2,09+/-2,40	15,64+/-14,37	0,72+/-0,57
1575	0,2+/-0,1	0,6+/-0,3	56,36	74,36	0,03+/-0,17	2,33+/-2,5	16,75+/-15,24	0,89+/-0,70
1280,4	0,2+/-0,1	0,7+/-0,3	58,76	75,66	0,029+/-0,17	2,39+/-2,62	16,83+/-15,25	0,90+/-0,74
1892,1	0,2+/-0,1	0,7+/-0,2	57,59	75,54	0,014+/-0,11	2,37+/-2,62	16,61+/-15,04	0,88+/-0,75
2253,8	0,2+/-0,1	0,8+/-0,2	60,94	77,66	0,043+/-0,21	2,46+/-2,61	16,89+/-15,37	0,91+/-0,73
1227,5	0,3+/-0,1	0,6+/-0,3	57,92	75,5	0,029+/-0,168	2,33+/-2,41	17,07+/-15,62	0,87+/-0,80
1038,2	0,4+/-0,1	0,6+/-0,3	60,38	77	0,057+/-0,293	2,30+/-2,58	16,40+/-15,14	0,84+/-0,71
887,5	0,5+/-0,1	0,6+/-0,3	62,5	78,2	0,043+/-0,27	2,12+/-2,29	16,54+/-15,4	0,88+/-0,76
1576,6	0,5+/-0,1	0,8+/-0,1	65	80,3	0.057+/-0,294	2,26+/-2,66	16,63+/-15,17	0,96+/-0,85
951,4	0,6+/-0,1	0,7+/-0,2	67,5	81,6	0,014+/-0,119	2,07+/-2,13	16,25+/-14,91	0,98+/-0,82
814,7	0,7+/-0,1	0,7+/-0,2	73,8	85,3	0.014+/-0,119	1,91+/-1,82	16,16+/-14,4	0,96+/-0,72
905,1	0,7+/-0,1	0,8+/-0,1	75,1	85,99	0,11+/-0,54	2+/-1,82	16,42+/-14,61	1,02+/-0,79

Table 1: *Comparison between multiple graphs and quality of the sequences generated*

variation of the graph, the number of unique sequences generated from minimum input is almost limitless.

6. Results & Discussion

We have implemented our procedural approach in *Matlab*© and tested it in a laptop with an i7-4720HQ and a NVidia GTX 970M. The landmarks sequences are imported in *Autodesk Maya 2011*© and applied to two 3D blendshaped face models using the direct manipulation technique [LA10] as described in sec. A. Nevertheless, the 3D animation results of the accompanying video should serve only as a rough approximation of how the final animation could look like, with better results achievable artistic help. We analyse facial motion graphs quantitatively in terms of compression, and by comparing the new animation against the original sequences. We additionally discuss how the proposed method relates with the work of Zhang et al. [ZSCS04] and with a traditional facial motion capture approach (sec. A). Accompanying video shows several samples generated using the proposed approach, from where fig. 6 was extracted. The video contains animations in two different blendshaped models that serve as practical examples where the motion graphs could be used. As more models are used, it becomes even harder to identify similarities in generated motions.

The motion graphs were created from 70 sequences of ~20 subjects of the CK/CK+ [KCT00, LCK*10] DB. We have compared the training samples, i.e. baseline, against the synthesized equivalent and found that, even for high compression ratios, the proposed technique is capable of synthesising motion similar to the original data. Compression ratio refers to the memory footprint gains compared to a graph with a node per pose of the DB. As a reference, this base graph has 1792 nodes. The results can be seen in table 1, whose columns are: 1) motion graph creation time; 2) and 3) contain the different inputs used in the compression optimization respectively for the stages of sample graph creation and merging; 4) and 5) show the compression ratios in regard to number of nodes and in terms of data, i.e. landmarks and numerical values associated to motion recovery; 6) duration difference and respective deviation, between each new sequence and its counterpart in the DB; 7) landmark distances, and deviation, between each sample pose and respective synthesized pose; 8) angle difference, and respective deviation, between instantaneous velocities calculated for the same poses as the previous column. The results are obtained using the dot product and converted to % for easier interpretation. A perfect match corresponds to 0% and 90° to 100%; 9) synthesis times, and their deviation, required to generate a new sequence.

This table provides different insights on how motion graph creation and animation synthesis behave. Regarding the first, data compression ratio falls typically within the boundaries defined by the user input, however it is near impossible to predict the actual values since this is highly dependent on the content of the DB. The nodes' compression value can be a "misleading" measure as it accounts for the nodes of all the region graphs. Therefore, these nodes easily surpass the number of nodes of the baseline for lower compression values. However, even with more nodes, similar landmarks have been merged into the same node, and as a result data compression already exists. Additionally, the structure representing the connections, which traditionally is a matrix, grows exponentially with the number of nodes of the graph. Therefore significant savings occur by creating a smaller matrix for each region graph instead of a matrix that connects all DB poses. The processes of creating and using the motion graphs are also subject to local minima, whose impact can be seen almost in all columns. An example is the case of the data compression of 85.3%, that recovers motions more similar to the DB than other lower compression cases. Here, the structure of the graph represents the core poses better, which leads to a motion recovery better than other structures with lower compression. Nevertheless, a lower data compression will generally represent the motion better and require more time to be created. This behaviour will be similar with larger DBs, i.e. more samples will lead to longer generation times, with the size of the graph following the desired compression.

Regarding the differences between original and synthesised sequences, the duration shows almost no loss of information, which is explained by the choice of storing the peak duration. Differences in the landmarks position have an error ~2-4 pixels, which is acceptable given the range of landmarks positions, ~270 pixels in the x axis, and ~380 pixels in the y axis. Finally, the velocity error ranges between 10 to 30%, which we consider acceptable given the achieved compression. An interesting consequence of our method is that for the lowest compression levels data recovery is still not

complete. This results from some information being lost in merged poses. This would be only possible with extremely low compression values, in which the extra data (sec. 4.4) would lead to more space being used than the original data. These results confirm the method significantly reduces the DB size while keeping the lost information to a minimum. Also relevant is the synthesis time that is reduced as the graph size increases. This occurs due to bigger graphs requiring less time to recover information since sigmoid fitting converges faster. Additionally, the path finding time for this type of graph and size is negligible. In larger graphs that originate from more comprehensive DBs, the path finding will start impacting the synthesis time, however we expect the results to still be acceptable. This results from Dijkstra's algorithm [Dij59] performing well in sparse graphs with a complexity of $\mathcal{O}(Elog(N))$, where E is the number of edges and N the number of nodes [CLRS09]. Which is the case of the created graphs due to the nature of facial poses that connect and are followed only by similar poses. With the present graph, synthesis performs near real-time and as a result we believe that porting this technique to a game engine would easily surpass this constraint.

The approach closer to ours is the face graph of Zhang et al. [ZSCS04]. In terms of the quality of motions, the results are similar with different issues, as seen in the accompanying video. Face graphs produce motions that occasionally have distinguishable phases, i.e. there is a mid expression, usually neutral, when going from expression A to B. Our approach sometimes produces distinguishable region timings, which would also occur in Zhang's [ZSCS04] approach if a graph per region would be created. Both can occasionally generate an exaggerated pose jump. The proposed method's main advantage is the compression, as a face graph is created by connecting all poses with all other. This indeed fully recovers the motions from the DB, however it also leads to a longer path finding, usually taking ~1.9 seconds. We have also compared our approach with a more traditional take on animation (sec. A). We markerlessly capture the performance of a person executing the same order of expression labels as the equivalent synthesised sequence. In this context, face capture tends to provide more expressive and natural results, with the actor making poses that better match the model shapes. This comes at the cost of noise that leads to strange motions. Our method produces considerably more well-defined motion that is smooth. However, this also leads to more stiff motion. Generation via motion graphs is also limited to the poses present in the DB, which does not occur with performance capture. On the other hand, our approach requires far less input to create an animation, eliminates the need for additional equipment and the actor. Performance capture and procedural animation share some challenges, such as pose alignment, however these are significantly different with distinct requirements and constraints. As a result, our goal with this comparison is just to provide a reference point on how both fare.

The chosen DB and its impact on the results also need be referred. The main reasons for the choice of CK/CK+ are presented in sec. 4, however it is less than ideal. It was not created for facial animation, the poses are sometimes too small to be applied to a model, leading to animations that are hard to recognize. The landmarks are either not enough or properly placed, and contain some jitter even after cleaning. It only has motions from neutral-to-peak, whose direct use would lead to a graph that only generates paths to peak expressions, i.e. dead-ends. This issue is "*bypassed*" by appending to the end of a sample, its own reversed copy. Nevertheless, facial behaviors have different onsets and offsets [ACR09], thus ideally the full behaviors should be used. Only having neutral-peak-neutral transitions proves that our method can learn the dynamics of transitions, but it does not mean these are the most correct ones.

The main limitations of the presented method include its heavy reliance on well defined peak expressions, which is not always the case, such as in subtle motions. More complex motions such as visual speech might require an extremely large graph. Our approach to align and reduce the effects of individuality, via Procrustes' analysis, is also not sufficient to completely remove all the effects of different proportions, and can sometimes introduce peculiar motions. High compression values can result in peak expressions of different labels merged in the same peak node. This translates into the character displaying the same pose for multiple expression labels. When recovering data, using sigmoid fitting to smooth the results can also lead to excessive removal of the individual variations, hence removing a layer of uniqueness. Facial behaviors also include not only facial expressions, but gaze and head movements as well, which cannot be represented with the current graph structure. Finally, while not a limitation, a seed sequence is randomly selected and used as the starting graph. This makes the graph slightly different each time it is generated.

7. Conclusion

In this paper, we proposed a procedural method that relies on motion graphs, created for each facial region, to represent and synthesize facial behaviors. This method can generate non-repetitive, on-the fly animations with minimum input, namely the expression label. The label is used to find destination nodes associated to most intense expressions. The path is calculated by minimizing the similarity metric between the source and destination nodes. Our method is capable of generating a limitless number of unique sequences, due to the choice of having separate motion graphs for each region, and by introducing small variations in the minimum distance path. The author controls how much information should be kept in the graphs, with the thresholds optimized to achieve the desired compression. This makes the specification of a motion graph considerably easier, and allows for motion graphs to better represent the DB data, as each sample had its own individual optimal threshold. Additionally, the proposed approach is capable of representing the DB in much smaller memory footprint, with a reduced loss of information. An existing graph is also easily extensible, since a new sample can just be merged into the graph as long as it follows the same landmarks configuration, and has the same reference pose.

Our priorities for the **future** include extending the motion graph to support gaze and head motions, which is required for attaining more realistic facial behaviors. In the same sense, we intend to explore how to represent more complex movements such as nodding, expressing pain and speech without increasing the complexity or the size of the graph exponentially. Also, facial region graphs are treated separately, which sometimes leads the disconnected region movements. Solving this is required to achieve more credible

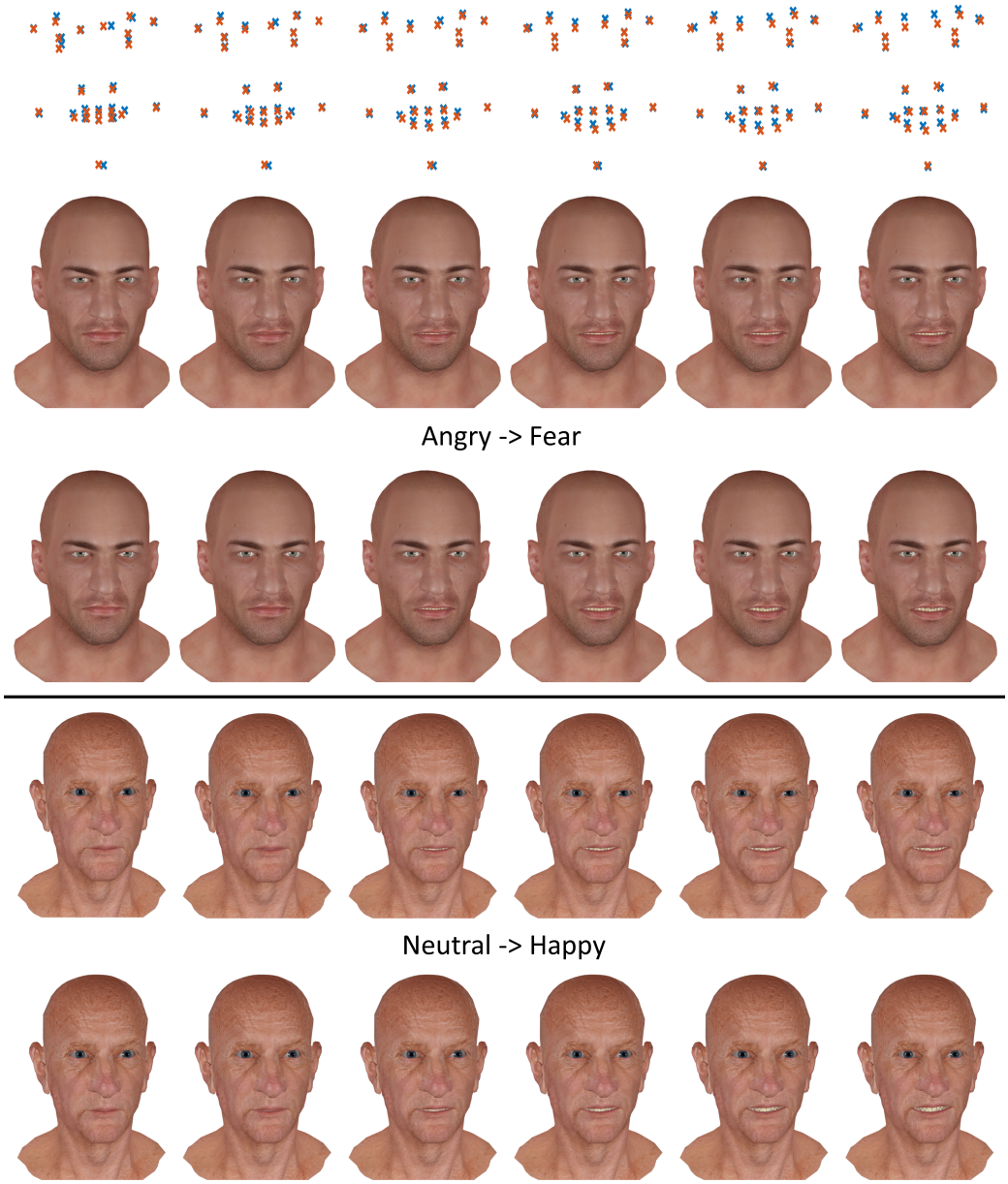

Figure 6: *Example sequences generated from applying the synthesised landmarks onto a 3D facial model. In the first row it is possible to observe the actual landmarks associated to the following two model sequences. All the poses were extracted from the accompanying videos.*

motions. The proposed method is particularly useful for animating non-playable characters (NPC) that interact with the player or between themselves, in crowds of both video-games and films. In a game, the characters can rely on behavior rules to trigger expression labels associated to the motion graph, thus creating a new animation on each interaction, as is shown in [SOC16]. In crowds, the same approach can be used, or as an alternative, the user can directly author the crowd behavior. In both cases, the required input is greatly reduced compared to both key-frame and performance-driven animation. Main characters of films and video-games can also be animated, at least on a basic level. The two main reasons for such being: the proposed method relies on a DB that may not have the desired motion expressed in the desired way, and, as the results still contain some errors, fine-tune is necessary. Motion graphs have a wide range of applications. and although their use is not a new idea, their application in facial animation remains under-exploited. Therefore, we hope to propel new research in this particular area.

8. Acknowledgements

We would like to thank *Faceware*© for allowing the use of the Victor 3D facial model.

References

[ACR09] AMBADAR Z., COHN J., REED L.: All smiles are not created equal: Morphology and timing of smiles perceived as amused, polite, and embarrassed/nervous. *Journal of Nonverbal Behavior 33*, 1 (2009), 17–34. 9

[AD07] ARYA A., DIPAOLA S.: Multispace behavioral model for face-based affective social agents. *J. Image Video Process. 2007*, 1 (Jan. 2007), 4–4. 2, 3

[Art] ARTEC: Artec 3d scanners. *http://www.artec3d.com*. 12

[ARV07] AMBERG B., ROMDHANI S., VETTER T.: Optimal step nonrigid icp algorithms for surface registration. *In Proc. of IEEE Computer Vision and Pattern Recognition (CVPR)* (2007). 12

[BHPN01] BUI T. D., HEYLEN D., POEL M., NIJHOLT A.: Generation of facial expressions from Emotion using a Fuzzy Rule Based System. *IN: Lecture Notes in Artificial Intelligence 2256* (2001), 83 – 94. 2

[BSG10] BIDARRA R., SCHAAP R., GOOSSENS K.: Growing on the inside: Soulful characters for video games. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on* (Aug 2010), pp. 337–344. 2, 3

[CLRS09] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to Algorithms*. MIT Press, 2009. 9

[CTGH12] CASAS D., TEJERA M., GUILLEMAUT J.-Y., HILTON A.: 4d parametric motion graphs for interactive animation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D '12, ACM, pp. 103–110. 2

[DDLT02] DEVILLERS F., DONIKIAN S., LAMARCHE F., TAILLE J.-F.: A programming environment for behavioural animation. *The Journal of Visualization and Computer Animation 13*, 5 (2002), 263–274. 3

[Dij59] DIJKSTRA E.: A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 1 (1959), 269–271. 2, 6, 9

[Gle08] GLEICHER M. L.: Graph-based motion synthesis: An annotated bibliography. In *ACM SIGGRAPH 2008 Classes* (2008), SIGGRAPH '08, pp. 49:1–49:11. 2

[GSKJ03] GLEICHER M., SHIN H. J., KOVAR L., JEPSEN A.: Snap-together motion: Assembling run-time animations. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics* (New York, NY, USA, 2003), I3D '03, ACM, pp. 181–188. 2

[HG07] HECK R., GLEICHER M.: Parametric motion graphs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 129–136. 2

[HTCH15] HUANG P., TEJERA M., COLLOMOSSE J., HILTON A.: Hybrid skeletal-surface motion graphs for character animation from 4d performance capture. *ACM Trans. Graph. 34*, 2 (Mar. 2015), 17:1–17:14. 2

[KCT00] KANADE T., COHN J., TIAN Y.: Comprehensive database for facial expression analysis. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on* (2000), pp. 46–53. 4, 8

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (2002), SIGGRAPH '02, pp. 473–482. 2

[Kin09] KING D. E.: Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research 10* (2009), 1755–1758. 12

[LA10] LEWIS J. P., ANJYO K.: Direct manipulation blendshapes. *IEEE Comput. Graph. Appl. 30*, 4 (July 2010), 42–50. 8, 11, 12

[LAR*14] LEWIS J. P., ANJYO K., RHEE T., ZHANG M., PIGHIN F., DENG Z.: Practice and theory of blendshape facial models. In *Eurographics STAR 2014* (2014), pp. 199–218. 11

[LCK*10] LUCEY P., COHN J., KANADE T., SARAGIH J., AMBADAR Z., MATTHEWS I.: The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on* (2010), pp. 94–101. 4, 8

[LCP*11] LUCEY P., COHN J., PRKACHIN K., SOLOMON P., MATTHEWS I.: Painful data: The unbc-mcmaster shoulder pain expression archive database. In *Automatic Face Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on* (2011), pp. 57–64. 4

[MBC01] MIZUGUCHI M., BUCHANAN J., CALVERT T.: Data driven motion transitions for interactive games. In *Eurographics 2001 Short Presentations* (2001), vol. 2, p. 6. 2

[MR76] MEHRABIAN A., RUSSELL J.: The three dimensions of emotional reaction. *Psychology Today 10*, 3 (1976), 57–61. 3

[OBP*12] ORVALHO V., BASTOS P., PARKE F., OLIVEIRA B., ALVAREZ X.: A facial rigging survey. In *in Proc. of the 33rd Annual Conference of the European Association for Computer Graphics - Eurographics* (2012), ACM, pp. 10–32. 3

[Orf96] ORFANIDIS S.: *Introduction to Signal Processing*. Englewood Cliffs, NJ, Prentice Hall, 1996. 2, 7

[Per97] PERLIN K.: Layered compositing of facial expression. In *ACM SIGGRAPH 97 Visual Proceedings: The art and interdisciplinary programs* (1997), ACM Press, pp. 226–227. 2, 3

[PG96] PERLIN K., GOLDBERG A.: Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of SIGGRAPH '96* (1996), ACM Press, pp. 205–216. 2

[SBR*14] SAGAR M., BULLIVANT D., ROBERTSON P., EFIMOV O., JAWED K., KALAROT R., WU T.: A neurobehavioural framework for autonomous animation of virtual human faces. In *SIGGRAPH Asia 2014 Autonomous Virtual Humans and Social Robot for Telepresence* (2014), SA '14, pp. 2:1–2:10. 2, 3

[SOC16] SERRA J., ORVALHO V., COSKER D.: Behavioural facial animation using motion graphs and mind maps. In *Proceedings of MIG* (New York, NY, USA, 2016), ACM. 10

[SP04] SUMNER R. W., POPOVIC J.: Deformation transfer for triangle meshes. *ACM Trans. Graph 23*, 3 (2004). 12

[XMLD07] XUE Y.-L., MAO X., LI Z., DIAO W.-H.: Modeling of layered fuzzy facial expression generation. 243–252. 2, 3

[ZSCS04] ZHANG L., SNAVELY N., CURLESS B., SEITZ S. M.: Spacetime faces: High resolution capture for modeling and animation. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 548–558. 2, 3, 8, 9

[ZYC*14] ZHANG X., YIN L., COHN J. F., CANAVAN S., REALE M., HOROWITZ A., LIU P., GIRARD J. M.: Bp4d-spontaneous: a high-resolution spontaneous 3d dynamic facial expression database. *Image and Vision Computing 32*, 10 (2014), 692 – 706. Best of Automatic Face and Gesture Recognition 2013. 4

Appendix A: Video Generation

We now describe the processes used to generate the animations in both *Victor* and *Oldman* facial models from *Faceware*© present in the accompanying videos. All sequences, with the exception of the performance capture, are generated using the approach described in sec. A. The facial capture sequences are generated using the approach described in sec. A.

Direct Manipulation of Blendshapes

Our method connects to the blendshape direct manipulation method [LA10]. The mapping for generating the animation is created with 14 landmarks associated to the manipulators via parent constraint. We manually re-scale the landmarks to keep them proportional to the model, using the neutral pose as a reference. These manipulators are placed on the surface of the model, and allow updating the underlying blendshape weights by using a "pin-and-drag" operation. Theoretically, the blendshape model is the summation of the linear vectors of the defined target shapes [LAR*14] (equation 4):

$$F = \sum_i b_i w_i \tag{4}$$

where F is the final face in vector form, which includes all vector positions of the face model in an arbitrary order of xyz, b_i is the vector of each blendshape target, and w_i is the corresponding weight. According to the original mathematical framework of [LA10], there is a direct relationship between the blendshape weights (w) and the moved manipulators. After all the manipulators are located and moved on the surface of the face, the mathematical framework keeps the resultant face model as close as possible to the moved manipulator. This relationship can be explained as $m = Bw$, where m is the moved manipulator, B is the blendshape matrix which includes all blendshape target vectors. Therefore, by following [LA10], a regularized least square form is employed to create interactive weight updates to demonstrate our visual results (equation 5),

$$\min_w \|Bw - m\|^2 + \alpha\|w\|^2 \tag{5}$$

where α is the regularization term which should be small number such as 0.0001. As a result of equation 5, our weight update equation is:

$$w = (B^T B + \alpha I)^{-1} B^T m \tag{6}$$

Equation 6 updates the facial poses during the movements of each manipulator. Besides, we enhance the framework by adding an interactive feature which enables to calculate equation 6, and demonstrate the corresponding face movements simultaneously. Therefore, after the manipulators and locators are applied, the animator can visually follow the face movements with the movements of the manipulators.

Face Capture

Our face capture results were generated using a monocular blendshape based tracking approach. We first obtain the 3D mesh representing the neutral face of the actor, scanned using a 3D scanner – Artec Eva [Art]. An existing template neutral mesh and respective topology is then deformed to match the scanned mesh using the non-rigid ICP algorithm of [ARV07] The deformation transfer approach of [SP04] is then used to automatically generate person specific blendshapes with an existing template of blendshapes as a basis. This provides the set of blendshapes with desired topology that we then use to solve for animation parameters.

For tracking the actor's performance, we capture the monocular video of the actor. We then track 68 distinctive landmark points on the face, on a frame by frame basis using the dlib library [Kin09]. From these, we choose landmarks equivalent to the ones used in the direct manipulation approach. These landmarks are then manually matched with points in the 3D mesh. Finally, we solve for the optimal blendshape weights for each frame by minimizing the following energy term:

$$E = \sum_{l=1}^{N} \|\Pi_Q(M(B_0 + \sum_{i=1}^{N_B} \alpha_i B_i)^{(v_l)}) - q^{(l)}\|^2$$

where:

- N is the number of landmarks
- Π_Q is the camera projection matrix
- M is the rigid transform from object space to camera coordinates
- B_0 is the neutral expression blendshape
- α_i is the weight associated with blendshape i
- B_i corresponds to the i-th blendshape
- v_l represents the vertex corresponding to landmark l
- $q^{(l)}$ represents the l-th 2D landmark point in the image

The per-frame landmark detection can induce noise in the result, so we smooth the obtained blendshape weights over the sequence using a moving average filter to obtain our final animation weights.